A Digital Library Architecture that Promotes Creation and Use of Modifiable Student Activities

David Yaron, D. Jeff Milton and Michael Karabinos
Department of Chemistry, Carnegie Mellon University, Pittsburgh, PA 15213

Abstract

This paper explores the definition and use of "linked active content", an approach to the creation and organization of online educational activities in digital libraries (http://ir.chem.cmu.edu/create). Education software has the benefit of supporting flexible interactive learning environments that include domain-specific simulations and interactive data visualization. However, current software engineering practices lead to monolithic library entries that are difficult to subdivide and repurpose. A review of educational development practices that have succeeded in supporting reuse reveals a common approach of "configuration as authoring", in which simulations can be configured for a variety of student activities. Linked active content builds on this approach by enabling authors to assemble a variety of configurable components into a student activity, and supporting message passing between these components. The resulting student activity is a collection of linked configuration files. Students browse through this "linked active content" as they perform the activity. Our premise is that by separating content from software, linked active content addresses a number of challenges associated with software collections. Separating simulations and visualizations from the components used to construct student interfaces, such as text boxes and dials, enables them to be more easily repurposed to varied educational settings. Separating content from software also supports better maintenance, since the software components can be updated without requiring modifications to the content files. Perhaps most importantly, learning object assembly tools such as our CreateStudio should allow a more diverse audience to participate in the authoring process, including especially those who have substantial classroom and pedagogical expertise but lack programming expertise. This paper will describe the functionality we’ve found necessary to support various pedagogical approaches, such as support for exploration of virtual worlds and for multimedia that places domain-specific simulations in motivating contexts.

Introduction

This paper describes our work in creating a technical infrastructure that will allow digital libraries to serve as collaboration spaces for the creation, modification and dissemination of engaging, varied learning activities. Our approach is designed to address some of the challenges of creating engaging online educational experiences. From a community building perspective, creation of these activities requires both technical and pedagogical expertise, a combination of expertise that few individuals in the community can be expected to possess. From a digital library collections perspective, the level of interactivity required for engaging activities can lead to monolithic chunks of software that are difficult to subdivide and repurpose to varied educational settings. Our premise is that these community building and collections challenges are intimately coupled, since they both relate to where current technology draws the line between software creation and content authoring. The location of this line, as defined by current web technologies, is such that far too much of the design and development process for interactive online content is in the domain of software creation, and therefore in the hands of programmers more than instructors. The CreateStudio architecture pushes this line to allow for more powerful and flexible content authoring, such that we may better utilize the expertise of those members of the community with curriculum development and classroom experience. Moreover, in redefining the line, we extract content from the software in a manner that leads to better organized collections of content that can be more easily modified to meet the needs of varied educational settings.

Multimedia authoring tools such as Macromedia Director, Flash etc. allow non-programmers to create interactive online content that involves media such as text, images, and animations. However, many goals of STEME education, such as those in inquiry learning, are best met through simulations and visualization tools that are not easily implemented via multimedia. For such cases, many users have adopted Java as a creation and delivery platform. The Java platform provides a full-featured programming language and a virtual machine/plug-in that allows Java applets to be distributed via the web and run within popular browsers. Thousands of applets have been created for educational purposes and these are catalogued on sites such as the educational
object economy (www.EOE.org), which lists 2600 applets, and Merlot (www.merlot.org). Yet despite the large number of these applets, and the reasonably long standing efforts to catalog and organize them, the large majority of these applets do not find significant use outside of the specific course/context for which they were developed (McGraw, 1999). The problem is that the entry in such collections is currently an executable piece of software, and this leads to content that is difficult to adapt to the diverse needs of the educational community. For a particular piece of software to be useful, there must be good alignment between the goals of the instructor and the goals of the programmer who created the content. Even if source code is available, and the changes required to make the content useful in their particular classroom are small, instructors do not typically have the time or expertise needed to implement them.

The dilemma is that domain-specific simulations and visualization tools necessitate a flexible creation environment such as that of Java or other programming environments but the cost of this flexibility is a level of complexity that both excludes non-programmers from the development process and leads to content that is difficult to modify and extend. The goal of the digital library architecture described here is to resolve this dilemma.

**Active Content**

One potential aspect of digital libraries is their ability to deliver content that is both interactive and engaging. For instance, earth science educators have found a number of high quality uses for earth science data in K-12 classrooms (www.delse.org, Manduca et al, 1999), (www.globe.gov, Avard et al, 2001). For example the GLOBE (Avard et al, 2001) project by provides instructors with a yearly teachers guide that includes protocols for in-class activities, including visualization and analysis of data from the GLOBE digital library. The Digital Library for Earth Science Education (DLESE) contains several resources that students can use to monitor an ecological system in real-time. Interactive online content is also available in other important domains, such as biology, see the Biology Labs On-Line Project (Desharnais et al., 2002) physics, and chemistry (see below). However, the creation of high quality active content remains challenging and expensive.

**Current approaches to creation of active content**

A number of projects have begun addressing issues surrounding creation of active content. For instance, ESCOT (Roschelle et al., 1999) and ESLATE (Birbilis et al., 2000) couple programming tools that make the programming process more efficient with a community development process. The technology components of these projects are based on the software component model of code-reuse. Both ESCOT and ESLATE provide Java components that provide functionality that is of general use for education. Both also provide a component assembly environment that is rooted in SUN’s JavaBeans component assembly model but with modifications/simplifications designed to make the approach more accessible and efficient for educational purposes. This programmer support is then used in service of a community development process that teams pedagogical experts with programmers to develop educational software. For instance, ESCOT initially forms its teams at a workshop where they learn the technology and meet team members. They then work together for 4 weeks, each member contributing about 40 hours, to produce a Java applet, many of which are distributed on the MathForum (Alejandre et al., 1994). This approach addresses the reusability issue at the programmer level, by creating re-useable software components, but not the curriculum level, since the outcome remains a completed Java applet that is monolithic and difficult to repurpose. The tight coupling between programmers and curriculum developers does address the issue of the varied expertise needed for creation of useful educational software; however, the resource requirements make it difficult to scale this development-team approach to the national level.
Data Viewer (IDV), with a set of small Java applets, and the VGEE probes that are aimed explicitly at education.

**Configuration as authoring**

The projects discussed above that have achieved loose coupling between programmer and instructor are from a variety of different domains and have different technical implementations, but they share the basic approach we will call configuration as authoring. The utility of this approach arises from breaking the creation process into two stages. In the traditional approach to Java software development, a programmer produces a completed Java applet for use by a student. The programmer has thus taken the software all the way to completion. While an instructor or curriculum developer can place this on a web page along with guidance and instructions to the student, they are not free to alter the applet or modify its student interface. In configuration as authoring, the programmer does not take the project all the way to the student. Rather, the programmer produces a more general tool that the instructor then configures in the process of constructing a student activity. This breaks the development process into two stages. In the first stage, a programmer produces a simulation or visualization tool that provides generally useful functionality for a particular content domain. In the second stage, an instructor configures the tool for a particular student activity and provides the student with guidance and context.ii

This two-step development process allows programmers and curriculum developers to contribute in a manner that best leverages their particular expertise. Since the programming occurs only in stage one, we can give the programmer the flexibility and power of a full-featured programming environment without having the complexity of such an environment serve to exclude non-programmers from the development process. Likewise, since the simulation and visualization tools produced in stage one are aimed at instructors and curriculum developers, rather than directly at students, we do not require the programmer to have extensive classroom experience or pedagogical expertise. The two-step development process also leads to instructor modifiable content, since an instructor can reconfigure the tool using an existing configuration as a starting point.

The two-stages of the configuration as authoring approach are somewhat coupled, since the tools produced in the first stage must have functionality useful for the curriculum development of the second stage. However, this coupling is much weaker than that of the “design team” model of ESCOT and ESLATE, where curriculum developers and programmers work directly together. This approach can therefore support the use of digital libraries as collaboration spaces that utilize the internet to facilitate collaborations that are remote in both location and time.

**Linked active content**

Our library architecture couples the two-step development process of configuration as authoring with the assembly model of component software. The result of the authoring process is a set of configuration files, each corresponding to a different software component. Components are allowed to pass messages between one another, and these appear as links between the different configuration files, similar to the links between files in a web site. It is important that these links, which capture a relationship between software objects, are stored in the configuration files themselves rather than being hard-coded into the software. This achieves a separation of content from software that makes the content much easier to manage and maintain in digital library collections.

Consider, for instance, the Java applet shown in Figure 1. Students use this rocket trajectory simulator in our Mission Critical Chemistry activity (Section ?) to determine the amount of fuel needed to reach Mars. In the traditional approach, this applet would be a single executable stored as a monolithic chunk in a digital library. In our approach, the applet is constructed from software components, (i) a trajectory simulator that contains the domain-specific knowledge and (ii) an extended image map viewer that is similar to the HTML image map but that supports extended linking (see Section ?). The upper left frame contains the simulator and the upper right and lower frames contain image map viewers. In constructing the applet, the curriculum author used CreateStudio, described below, to draw out the three frames and load the respective software objects into the frames. The author then configured these components. The simulator is configured by specifying parameters such as the number of planets to show and various aspects of the rocket. The image maps are configured by loading the images shown in Figure 1, drawing hot spots over the appropriate regions of the image and specifying the extended links to be associated with these hot spots. In the upper right panel, the hotspots pass the launch and reset messages to the simulation. In the lower panel, the hotspots are editable hotspots that pass numbers entered by the student to the simulation. The result of the authoring process is three configuration/content files, which contain within them the appropriate link information. These files also contain pointers (or URL's) to their associated software objects.
Note that the software is stored independently of the content files, and a single piece of software can be used many times within and between applets. This has a number of collection advantages. From an organizational perspective, the content files can be tagged with metadata that supports multiple granularities in the discovery process. From a maintenance perspective, the software can be more easily updated to fix defects or to maintain compatibility with platform/operating system updates. Updating a software object in one location with compatibility with platform/operating system updates.

The software object in one location can be more easily updated to fix defects or to maintain compatibility with platform/operating system updates. Updating a software object in one location with compatibility with platform/operating system updates.

The CreateStudio authoring tool

From a curriculum developer's perspective

CreateStudio is an authoring tool that supports the creation of linked active content. The user interface is designed to capitalize on users familiarity with web content and web development tools such as Microsoft FrontPage. For instance, the frame concept of standard web development is extended to allow different frames to hold different software components, as in Figure 1. The user interface, shown in Figure 2, has three main panels.

- The File panel, on the left, holds the files which store the content and configuration for a learning object. The names of the files are displayed in a list, similar to the way existing files are listed in other programs such as FrontPage.
- The panel on the right is the main work area where users build frames and create content. The frames in CreateStudio hold different objects and can be linked to pass information between them, much like the way frames on websites work.
- The Tool panel is in the middle of the CreateStudio window. These tools change, depending on the content of the active frame. For example, if a simulation is loaded into a frame, the Tool panel will remove any existing tools and replace them with the tools used to configure the current simulation.

An authoring process begins by laying out empty frames of a desired size and shape, which will eventually hold the learning objects. To create a new frame, the user draws a box with the mouse in the main work area. The author can give the frames names so that they can receive messages as part of the extended linking mechanism. We call a collection of frames a frameset.

Once a frame is created and named, the user adds content by selecting that frame and loading an editor. This is done through the Create it item in the menu bar, which launches a dialog box that presents the user with a list of available learning objects (Figure 3). If the user selects, for example, QuickTimeVR, the viewer for QuickTimeVR images is loaded into the frame, and the toolbar in the center panel of Figure 2 is replaced with tools appropriate for this type of content. There are, in general, two different types of tools, and these are available on the “edit” and “collections” tabs shown in the circular insert for Figure 2. The “edit” tools allow one to configure the learning object and edit its content. The “collections” tools allow the user to load appropriate content into the viewer (see Section [portal]).

The user can switch between editing the different frames by clicking on the title bar of the frame. The toolbar in the middle panel changes according to the content of each frame. Users can easily create new content for a frame by saving the current content and loading a new editor into that frame. Linking the content of the frames enables the content in one frame to control the events in another frame (see Section [links]).

To allow for an iterative design process, and to support easy modifications to existing content, users can switch CreateStudio between preview and edit mode at any point. In Preview mode, the work area panel on the right of Figure 2 shows the application just as it would be seen by a student. The author can then step through the activity, switching to edit mode to make changes as needed. It is this feature that makes the active content easy to modify and repurpose to different educational contexts. When an instructor finds a student activity in a library that is “almost” what they want, they can load the content into CreateStudio, step through the activity in preview mode until they reach the section they would like to change, switch to edit mode to make the changes, and then republish the content back to the library.

Designing an interactive project in CreateStudio is analogous to using a flexible XML editor. For projects involving simulations, the user can think of CreateStudio as an editor that allows him or her to configure various parameters for, say, a combustion engine and save this to an XML file. The process of the authoring process is a set of linked XML files analogous to a website. As the user moves through the content, different pages are loaded into various frames. The difference here is that each file specifies its software viewer. This content also has extended links embedded within it.

From a programmer's perspective

CreateStudio is designed to make it easy for a programmer to add their own software components. This allows programmers to focus on the domain specific functionality, such as a simulation or visualization tool. By using CreateStudio, they gain the benefits of allowing authors to couple their component to a growing set of...
components including those supporting multimedia and construction of student interfaces. The architecture is intentionally designed to be extensible such that the programmer can replace any component, including the layout and multimedia components, if they are so inclined (see Section ?). This means there is very little loss of freedom associated with opting into the environment; a programmer can use those components they like and develop or replace components as desired.

To build an object into the CreateStudio framework, a programmer can provide the following functionality:

- A file format for the desired content, including an XML file with tags that point to the viewer and authoring tool.
- A viewer that is loaded by CreateViewer to display the content. For the viewer to accept messages, it must register its objects with the NameRegistry of Section ??.
- An editor that is loaded by CreateStudio to allow editing of the content. This will ideally take the form of a tool panel that appears in the middle panel of CreateStudio in Figure 2, but can also be a separate applet or application.
- A set of tools to appear on the Collections panel of Figure 2, to allow searching and loading appropriate content. For example an mpeg movie viewer would need access to a collection for loading mpeg content. This is only required if the available tool objects are not sufficient.
- A custom link parser, if the general purpose parser of Section ? does not provide sufficient functionality.

CreateViewer

Once building and editing in CreateStudio is complete the curriculum developer/instructor then publishes the applet/application. Currently there are three ways to publish.

- As a Java application to be run from a user’s hard drive or a CD-ROM. This is useful for situations where fast access to the internet is not available in the classroom. The instructors can utilize digital libraries to create student activities, and distribute these to their students in a manner that does not require internet access. Since the CD-ROM may contain its own Java virtual machine and associated software, this approach also isolates the application from a missing or poorly configured version of Java on the student’s computer.
- As a Java applet, along with an HTML page that includes the applet and can be placed on the user’s web server. Other projects are addressing the issue of supporting user-created content in digital libraries. For instance, a digital library service called the Instructional Architect (IA) project (Recker et al, 2001) allows users to publish content to an IA web server and then distribute the URL assigned by IA to their class.

The publishing process couples the content files and software objects with CreateViewer, which serves as a browser through the linked active content. Much of the functionality required for this browsing is within the Java virtual machine, and so the functionality required in CreateViewer is kept small and simple. CreateViewer initializes a Java application or applet and creates the graphical panel on which all visual learning objects will be displayed. It is responsible only for this top level panel. The frameset viewer handles the division of this panel into frames that can hold other content. The dynamic loading properties of the viewer allows other developers to provide alternative frame and screen handling systems, such as floating windows or tabbed panes. CreateViewer is also responsible for loading the link manager that provides the information-bus for the linking mechanism. Upon viewer activation the configuration files are parsed and links are processed. A naming registry manages the current state of the loaded viewers. When a link is subsequently activated upon student/user interaction, the link manager determines the appropriate learning object, uses the introspection ability of Java to query the loaded software objects to determine the types of links/messages they will accept, and passes on the message as appropriate.

CreateStudio as a creation portal for digital libraries

CreateStudio can serve as a portal for the creation and modifications of student activities. Integration falls into three general categories.

Discovery of learning objects Currently, when a user selects the “Create It” menu item in CreateStudio, they are given the dialog box of Figure 3 that lists the learning objects available on their local hard drive. Integration
into a digital library would replace this with a list of software objects in the library collection.

**Content discovery** While authoring content for a particular learning object, the “Collections” tab (Figure 4) allows user to discover materials appropriate for that learning object. Currently, this tab shows files on the local hard drive, but this could be replaced with search engines into library collections. These search engines would discover only content that is appropriate for the specific viewer. For instance, images for an image map (Acton, 2001), molecular structures for a molecular structure viewer (Berman et al, 2000), earth science data for a data visualization tools (Domenico et al, 2002). For simulations and other tools that implement the “configuration as authoring” paradigm, the configuration files may be tagged such that they can be discovered across multiple libraries and collections. Our current plan is to implement this through the mime type in Dublin Core (Weibel, 1997). The mime type will be constructed in a manner that guarantees uniqueness (one such approach is to combine the name of the learning object with a unique identifier for the source of the learning object, such as a registered domain name of the author.)

**Extended Linking**

Many of the benefits of our architecture results from the clear separation between software and content. This is achieved by storing the content in configuration files for the various software objects, and embedding links in the configuration files that capture the relationships between these software objects. These links pass messages between software objects. This mode of interobject communication is less general than that supported in general Java Bean assembly environments such as JBuilder (Borland Software Corporation, 1994), VisualAge for Java (International Business Machines, 1999), or Visual Cafe (WebGain Corporation, 1998). However, these tools are designed for software developers and so are not easily used by curriculum developers with out programming expertise. In particular, Java Bean assembly tools are built on top of Java compilers, and the errors issued by the compiler require considerable programming expertise to understand. In our architecture, communication between objects is done via message passing and errors appear as broken links, which are familiar to authors with web experience. In HTML, tools are available that automatically locate broken links. We are investigating the extent to which broken links may be automatically detected in linked active content.

The ability of this message passing approach to support various forms of active content is one of the primary research issues in this project, and this section discusses both our current implementation and what we have learned from using this approach to construct student activities.

**Linking architecture**

CreateViewer must coordinate the passing of messages between learning objects. To make this maximally flexible, we allow programmers to add their own parsers to the environment. This is done through a “Chain of Responsibility” design pattern (Gamma, 1994).

CreateViewer is at the top of the chain, and it examines the link to determine the appropriate parser. The parser is specified by the text preceding the first occurrence of “://”, with all text after the first occurrence of “://” being passed to the parser. A link then has the form

A://B

where A is a text string that specifies the parser and B is a text string that will be passed to this parser. The parsers are dynamically loaded Java class files. On startup, CreateViewer currently reads an initialization file that lists the names and ARLs (Application Resource Locators) of the parsers. (In integrating CreateStudio into specific digital libraries, it will most likely be advantageous to replace this mechanism of locating link parsers.)

To support dispatching of messages from the parser, CreateViewer keeps a NameRegistry that holds the names (as character strings) of Java objects that are able to accept messages, along with Java references (pointers) to these objects. Learning objects that are willing to accept messages must register themselves with the NameRegistry.

One link parser is that for standard web links of the form http://location, which loads an HTML page into a text window. This makes our extended links compatible with standard web links. The range of HTML constructs supported is dependent on the particular text window. Since we are currently using the HTML browser component provided in the Java Development Kit, the supported features are limited to those supported by this component (Sun Microsystems, 2002).

Java’s support for introspection, or the ability to perform a run-time analysis of a particular object, allows for the construction of generalized link parsers for Java software objects. Using introspection, a Java program may query a dynamically loaded learning object for a list of its methods (i.e. subroutines) and the data required by these methods. A general parser based on introspection can support links of the form

pars://target.action(data)

where target is the name of an object stored in the NameRegistry, action is the name of the method to be called on this object, and data is the parameters to be passed to this method. We are currently working on a
dialogue-driven authoring tool for this type of link, which will insulate the curriculum author from the link syntax. The tool will first present the author with a list of available targets, then present a list of the actions supported by the selected target, and finally query the author for the appropriate data.

Examples

When a programmer adds a software component such as a simulation or visualization tool to create studio, an important design consideration is the type of the messages the component with accept. These messages provide the functionality that the curriculum developer will use in constructing student activities. An advantage of this two stage development process is that the programmer can provide a component with a rich set of features. In a single stage development process, where the programmer creates an application for direct use by the student, the degree of functionality is typically limited by the need to keep the interface sufficiently simple that it does not confuse students. In the two stage development model, the curriculum developer uses CreateStudio to create a student interface that is appropriate for the educational context, for instance via control panels such as those of Figure 1. This interface can change between and even within an application by simply loading new control panels into the appropriate frames.

We have created a series of cross-domain components that support multimedia authoring with text, images, video, and quicktimeVR. The messages accepted by these components are fairly straightforward and simply load and manipulate the content. The frameset component has a link parser named “fset” that handles the arrangement and content of frames such as those of Figure 1. This accepts messages of the form fset://frame1.load(file1), where frame1 is the name assigned to the frame by the author and file1 is a content file. On loading, the software component associated with file1 will be loaded into the frame and its content will be displayed. The frameset component also accepts messages that replace the entire frameset with a new set of frames and associated content files. This allows authors to change the layout as a student moves through an activity. The frameset named start is loaded when the applet or application begins, and so this frameset plays a role that is analogous to index.html or default.htm in web development. As was previously mentioned the link editing mechanism will be replaced with a simple dialog driven link generator.

We’ve found that the image map construct can be extended to support creation of simple user interfaces to simulations. For instance, in Figure 1 control panels are constructed from images that contain features resembling buttons and text boxes. Our image map configuration tools are then used to draw hot spots over these images and attach links that pass messages. For instance the buttons of Figure 1 pass the messages load and reset. Bubble help can also be attached to these hotspots with the image map configuration tools. The image map also supports editable hot spots, which take text from the user and pass it to the simulation. This is shown in the lower panel of Figure 1. An even simpler form of control panel can be constructed with text links, for example, by using the text component of CreateStudio with text “click here” to launch the rocket and click here to reset”, where the underlined here’s are extended links that pass the appropriate message to the simulation. Slider bars and other interface components that are allow a user to set parameters of a simulation can also be supported within the current message passing mode of inter-object communication. To support interface components such as gauges or displays that monitor variables in a simulation, the inter-object communication would need to be extended to support querying an object for instance by calling a subroutine that returns data.

In some simulations, a little functionality can go a long way. For instance, our chemistry virtual laboratory accepts messages that cause solutions to appear on the laboratory workbench. A hotspot on an image, for instance over a glass containing a clear liquid, can pass a “load-solution” message to the virtual lab, giving the user the impression that they are collecting chemical samples. This also illustrates the power of coupling multimedia to simulations. A principle challenge of chemical education is helping students see the connection between chemistry and the real world. By coupling multimedia with the virtual lab, we can place chemistry activities in motivating real world scenarios. In our Mixed Reception activity, students watch a video of a party at which a chemistry graduate student dies from his peanut allergy, despite the fact that he is taking a drug for this allergy. Students then interview suspects, through short video clips, and collect chemical samples from the crime scene and other locations. The chemical evidence reveals an interaction between the allergy drug and a spider bite antitoxin that the graduate student is investigating as part of his thesis research. The students must then determine how the antitoxin got into the graduate students blood stream.

State and long term persistence

The Mixed Reception activity discussed above is the most complex application that we have constructed using CreateStudio, and allowed us to explore various aspects of the construction of large-scale activities. The activity combines about 40 minutes of video with quicktimeVR images of the crimescene and three other locations where
students collect chemical samples and other forms of evidence. The need to keep track of what evidence the student has collected is an example of the need to save information about the state of the student during the activity. Since the activity is lengthy, it would be convenient for the student to be able to save his current state to a file, and continue the activity later (persistence). An interesting issue is the extent to which software components can be created that support state and persistence in a manner that is sufficiently general that they can be useful in many contexts.

For Mixed Reception, we developed a backpack component that keeps track of what evidence the student has collected and that can store and retrieve this information from a file. This component appears in a window on the screen that contains icons indicating what evidence has been collected. The authoring tool used to configure this component allows the author to enter a list of items along with their icons. The backpack is initially empty and accepts messages that indicate what items have been collected, at which point it displays the appropriate icons. The author can also indicate a message to be sent in response to a click on an icon. This supports a common mode of game design, in which a user collects items and these items give them the ability to do certain things. For instance, in Mixed Reception, students must go to the coroner to get the autopsy report. This is implemented by sending a message to the backpack indicating that the autopsy report was obtained. Clicking on the icon of the autopsy report then loads the full text of the report into a frame.

Another example of persistence is collecting information on student interaction with an activity. This can be done by a software component that accepts messages indicating what the student has done. In Mixed Reception, we plan to add such a component and pass messages when the student makes relevant decisions. The IMMEX project (Stevens, 2001) has found analysis of such traces can allow one to group students into clusters that correspond to distinct problem solving strategies. Such a component can also be extended to provide help or guidance in response to certain pathways through an activity.

**Summary and Comments**

The architecture discussed here combines configuration as authoring approach with component software technology to achieve a new approach for both the creation and management of software in digital libraries. The creation advantages result from a two-stage development process in which programmers create software components that are then configured and assembled by authors without programming expertise. This loose coupling between programmer and instructor/curriculum developers supports a community development process in which the digital library serves as collaboration point between the programmers, who create components for use by instructors, and curriculum developers, who create student activities assembled from these components. This approach also supports an iterative development process, since it is easy for an author to load an existing activity into CreateStudio, step through the activity in preview mode to the point they would like to change, switch to edit mode to make the desired modifications, and publish the content back into the digital library.

The collections management advantages result from the clear separation between software and content, via linked active content. Due to this separation, a software component can be easily updated or changed and have this change reflected in all activities that use that component. Linked active content also reduces the grain size from a single monolithic Java applet or application to a set of linked configuration files that can be managed at a level of granularity that is set by the nature of the content rather than by technical issues relating to the content type.

An important aspect of the approach is that it places very few restrictions on the programmer. As discussed in Section ?, the programmer is free to either use the provided functionality or replace it with their own implementation. The architecture does not require a significant change in the mechanics of software development, rather it makes the component software techniques required in an object oriented programming language such as Java compatible with digital library collections. It does however, require a change in the target audience for the programmer. Rather than create completed student activities, the target is to provide authors with components that provided needed functionalities. We do not anticipate this to be a difficult shift, since the creation of components does not preclude the programmer from also developing student activities based on their components.

We believe CreateStudio provides a powerful approach to the creation of active student content. However, its success relies on its adoption by a community of programmers and authors interested in creating interactive student activities. The large number of Java applets for education already present on the web, along with the large number of instructor created web pages, suggests that such a community exists. Our goal is to provide this community with the tools they need to populate educational digital libraries with content that, through iterative community-based design, is of high
quality and connected to the needs of instructors and students.

References:


Figures:

Figure 1: Java Applet used by students in Mission Critical Chemistry to determine the amount of fuel needed to reach Mars. This illustrates the use of image maps and extended links to allow a non-programmer to attach a user interface to a simulation.

Figure 2: In the CreateStudio authoring tool, a curriculum developer selects software components via the Create It menu, arranges them on the developer workbench, and configures them using the tools on the tools panel. The authoring process generates project files that together form a set of linked active content.
Figure 3: The CreateStudio authoring tool being used to construct an activity. The dialog box lists available software components that can be placed in frames on the workbench.

Figure 4: The CreateStudio authoring tool with an expanded view of the Tools and Collections tabs.
For instance, addition of this functionality to Macromedia tools is done through LINGO, which is essentially a programming language.

Instructional uses of mathematical software such as MathCad, Mathematica, Maple, and Matlab, and SAS can also be pulled under the umbrella of configuration as authoring, although the target audience for the programming task of stage one extends well beyond education.